Component Stability in Low-Space Massively Parallel Computation

Artur Czumaj
University of Warwick
Coventry, United Kingdom
A.Czumaj@warwick.ac.uk

Peter Davies
IST Austria
Klosterneuburg, Austria
Peter.Davies@ist.ac.at

Merav Parter
Weizmann Institute of Science
Rehovot, Israel
Merav.Parter@weizmann.ac.il

ABSTRACT

In this paper, we study the power and limitations of component-stable algorithms in the low-space model of *Massively Parallel Computation (MPC)*. Recently Ghaffari, Kuhn and Uitto (FOCS 2019) introduced the class of *component-stable* low-space MPC algorithms, which are, informally, defined as algorithms for which the outputs reported by the nodes in different connected components are required to be independent. This very natural notion was introduced to capture most (if not all) of the known efficient MPC algorithms to date, and it was the first general class of MPC algorithms for which one can show non-trivial conditional lower bounds. In this paper we enhance the framework of component-stable algorithms and investigate its effect on the complexity of randomized and deterministic low-space MPC. Our key contributions include:

- We revise and formalize the lifting approach of Ghaffari, Kuhn and Uitto. This requires a very delicate amendment of the notion of component stability, which allows us to fill in gaps in the earlier arguments.
- We also extend the framework to obtain conditional lower bounds for deterministic algorithms and fine-grained lower bounds that depend on the maximum degree Δ.
- We demonstrate a collection of natural graph problems for which non-component-stable algorithms break the conditional lower bound obtained for component-stable algorithms. This implies that, for both deterministic and randomized algorithms, component-stable algorithms are conditionally weaker than the non-component-stable ones.

Altogether our results imply that component-stability might limit the computational power of the low-space MPC model, paving the way for improved upper bounds that escape the conditional lower bound setting of Ghaffari, Kuhn, and Uitto.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; • **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Pseudorandomness and derandomization**.

PODC '21, July 26–30, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8548-0/21/07…$15.00
https://doi.org/10.1145/3465084.3467903

KEYWORDS

Massively Parallel Computation; Lower bounds; Derandomization

ACM Reference Format:
Artur Czumaj, Peter Davies, and Merav Parter. 2021. Component Stability in Low-Space Massively Parallel Computation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3465084.3467903

For brevity, proofs and some results are deferred to the full version of this paper, available at https://arxiv.org/abs/2106.01880.

ACKNOWLEDGMENTS

This work is partially supported by a Weizmann-UK Making Connections Grant, the Centre for Discrete Mathematics and its Applications (DIMAP), IBM Faculty Award, EPSRC award EP/V01305X/1, European Research Council (ERC) Grant No. 949083, the Minerva foundation with funding from the Federal German Ministry for Education and Research No. 713238, and the European Union's Horizon 2020 programme under the Marie Skłodowska-Curie grant agreement No 754411.

1 INTRODUCTION

The central goal of this paper is to advance our understanding of the computational power of low-space algorithms in the *Massively Parallel Computation (MPC)* model. Our main focus is on the notion of *component-stable* low-space MPC algorithms introduced recently by Ghaffari, Kuhn and Uitto [18] as the first general class of MPC algorithms for which non-trivial conditional lower bounds can be obtained. Roughly speaking, in this class of algorithms the output of nodes in different connected components are required to be independent. While this definition has been introduced to capture most (if not all) of the known MPC algorithms to date, and the notion of component-stable algorithms seems quite natural and unlimited, we demonstrate its effect on the complexity of randomized and deterministic low-space MPC. Our main finding is that the notion of component-stability as defined in [18] is rather fragile and needs to be studied with care, leading us to a revision of this framework to make it robust. Our amended framework of component-stable algorithms allows us to fill in gaps in the earlier arguments and make it more applicable. In particular, the revised setup enables us to extend the framework of (conditional) lower bounds from [18] for component-stable randomized algorithms relating LOCAL algorithms and low-space MPC algorithms: we demonstrate that it can be parameterized with respect to the maximum graph degree Δ and holds also for deterministic algorithms, thereby making the framework more broadly applicable and proving for the first time

a host of conditional lower bounds for a number of deterministic low-space component-stable MPC algorithms.

Next, we will show that for some natural problems there are low-space *component-unstable* MPC algorithms (both randomized and deterministic) that are significantly more powerful than their component-stable counterparts. So, rather than being a technical triviality, component-stability is in fact a significant restriction on the power of the low-space MPC model.

Background. The rapid growth of massively parallel computation frameworks, such as MapReduce [14], Hadoop [33], Dryad [22], or Spark [34] resulted in the need of active research for understanding the computational power of such systems. The *Massively Parallel Computation (MPC)* model, first introduced by Karloff et al. [23] (and later refined in [1, 5, 21]) has became the standard theoretical model of algorithmic study, as it provides a clean abstraction of these frameworks. Over the past years, this model has been receiving a major amount of interest by several independent communities in theory and beyond. In comparison to the classical PRAM model, the MPC model allows for a lot of local computation (in principle, unbounded) and enabled it to capture a more "coarse–grained" and meaningful aspect of parallelism (see, e.g., [2, 6, 13, 16, 20]).

In the MPC model, there are M machines and each of them has S words of local space at its disposal. Initially, each machine receives its share of the input. In the context of graph problems where the input is a collection V of nodes and E of edges, $|V| = n$, $|E| = m$, the input is arbitrarily distributed among the machines (and so $S \cdot M \geq n + m$). In this model, the computation proceeds in synchronous *rounds* in which each machine processes its local data and performs an arbitrary local computation. At the end of each round, machines exchange messages. Each message is sent only to a single machine specified by the sender. All messages sent and received by each machine in each round, as well as the output, have to fit into machines' local space S.

Our focus in this paper is on the *low-space* setting where the local space of each machine is *strongly sublinear* in the number of nodes, i.e., $S = n^{\phi}$ for some $\phi \in (0, 1)$. Our lower bounds will be against algorithms with any polynomial number of machines (i.e., $M = \text{poly}(n)$), while our upper bounds will generally use at most $O(m + n^{1+\phi})$ global space (i.e., $M = O(\frac{m}{n} + n^{\phi})$).

The low-space regime is particularly challenging due to the fact that a node's edges cannot necessarily be stored on a single machine, but rather are scattered over several machines. Nevertheless, for many classical graph problems, poly$(\log n)$-round algorithms can be obtained, and recently we have also seen even *sublogarithmic* solutions. Ghaffari and Uitto [20] (see also [29]) presented a randomized graph sparsification technique resulting in $\widetilde{O}(\sqrt{\log \Delta})$ round algorithms for maximal matching and MIS, where Δ is the maximum degree. This should be compared, for example, with maximal matching algorithms with significantly more local space: Lattanzi et al. [26] presented an $O(1/\varepsilon)$-round randomized algorithm using $O(n^{1+\varepsilon})$ local space and Behnezhad et al. [6] gave an $O(\log \log n)$-round randomized algorithm using $O(n)$ local space (see also [3, 13, 16]). For the problem of $(\Delta + 1)$-vertex coloring Chang et al. [10] showed a randomized low-space MPC algorithm that works in $O(\log \log \log n)$ rounds, when combined with the network decomposition result of Rohzoň and Ghaffari [31].

While we have seen some major advances in the design of low-space MPC algorithms, no (unconditional) hardness results are known for any of the above problems in the low-space MPC setting. A seminal work by Roughgarden et al. [30] provides an explanation for the lack of such lower bound results. They showed that obtaining any unconditional lower bound in the low-space MPC (for algorithms with an arbitrarily polynomial number of machines) setting ultimately leads to a breakthrough result in circuit complexity, namely that $\mathsf{NC}^1 \subsetneq \mathsf{P}$. This work has opened up a new avenue towards proving *conditional* hardness results that are based on the widely believed *connectivity conjecture*. This conjecture (extensively used in our current paper) states that there is no $o(\log n)$-round (randomized) low-space MPC algorithm (even using any polynomial global space) for distinguishing between the input graph G being an n-length cycle and two $\frac{n}{2}$-length cycles.

The first conditional lower bounds in the low-space MPC setting were presented by a recent insightful paper of Ghaffari, Kuhn and Uitto [18]. This work provides a collection of conditional hardness results for classical local problems by drawing a new connection between the round complexity of a given problem in the LOCAL model [27], and its corresponding complexity in the low-space MPC model. Unlike the low-space MPC setting, for the LOCAL model, arguably one of the most extensively studied model in distributed computing, there is a rich collection of (unconditional) lower bound results. To enjoy these LOCAL lower bound results in our context, [18] presented a quite general technique that for many graph problems translates an $\Omega(r)$-round LOCAL lower bound (with an additional requirement of using shared randomness) into an $\Omega(\log r)$-round lower bound in the low-space MPC model *conditioned on the connectivity conjecture*. This beautiful lifting argument is surprisingly quite general, capturing the classical lower bounds for problems like MIS, maximal matching [25], LLL (Lovász Local Lemma), and sinkless orientation [7]. For example, one very strong implication of their technique is that conditioned on the connectivity conjecture, it shows that there is no randomized low-space MPC algorithm for (even approximate) maximal matching or MIS problems using $o(\log \log n)$ rounds.

The framework of Ghaffari, Kuhn and Uitto [18] has one main caveat, which at first glance appears to be quite negligible, a mere technicality. Their conditional lower bounds do not hold for *any* algorithms but rather only for the special class of *component-stable* MPC algorithms. The key property of these algorithms is that the output of the nodes in one connected component is independent of other components. More formally, in component-stable algorithms, the output of each node v is allowed to depend (deterministically) only on the node v itself, the initial distribution, the ID assignment of the connected component of v, and on the shared randomness. The class of component-stable algorithms is indeed quite natural, and at the time of publication of [18], it appeared to capture most, if not all, existing MPC algorithms, as explicitly noted by the authors:

> [18] *To the best of our knowledge, all known algorithms in the literature are component-stable or can easily be made component-stable with no asymptotic increase in the round complexity.*

In this view, it appeared that the restriction to component-stable algorithms is no more than a minor technicality rather than an actual limitation on the low-space MPC model.

The first indication that component-stability might actually matter was provided by recent works [11, 12], which present deterministic low-space *component-unstable* MPC algorithms for several classic graph problems, even though the validity of solutions to these problems depends only on local information. Specifically, by derandomizing a basic graph sparsification technique, one can obtain $O(\log \Delta + \log\log n)$-round deterministic low-space component-unstable MPC algorithms for MIS, maximal matching, and $(\Delta+1)$-coloring. A key ingredient of these algorithms is a global agreement on a logarithmic length seed, to be used by all nodes in order to simulate their randomized decisions. This global seed selection involves coordination between all the nodes, regardless of their components, thus yielding component-unstable algorithms. The component-instability here seems to be inherent to the derandomization technique, and it is unclear whether component-stable methods could perform equally well.

1.1 Our aims

In this paper *we thoroughly investigate the concept of component-stability and its impact on randomized and deterministic low-space MPC algorithms.* Upon examining the notion of component-stability in detail and after attempts to broaden its applications, it becomes apparent that the concept is highly sensitive to the exact definition used, and that one must be very careful in specifying on what information the outputs of component-stable algorithms may depend. For example, we must specify whether we allow component-stable algorithms' outputs to depend on the input size n, and we find that either choice here holds problematic implications for the current lower bounds and for the analysis due to Ghaffari et al. [18].

This raises the first main question of our work:

> Question 1. *Can we revise the lifting framework and amend the definition of component-stability which both captures a wide array of algorithms, and also allows us to prove robust lower bounds?*

Having fixed such a definition, we ask to what extent component-stability restricts MPC algorithms, and whether the concept is indeed a technicality or actually a significant limitation. The lifting arguments of [18] are designed for randomized algorithms, which raises the following question:

> Question 2. *Does component-instability help for obtaining improved randomized low-space MPC algorithms for graph problems? Is there any* separation between randomized component-stable *and* component-unstable *MPC algorithms?*

We then turn to consider the impact of component-stability on *deterministic* low-space MPC algorithms. Since the recent derandomization technique of [11, 12] leads to inherently component-unstable algorithms, we ask:

> Question 3. *Does component-instability help for obtaining improved deterministic low-space MPC algorithms for graph problems? Is there any* separation between deterministic component-stable *and* component-unstable *MPC algorithms?*

Understanding the gap between randomized and deterministic solutions is one of the most fundamental and long-standing questions in graph algorithms. In a very related context, the last few years provided a sequences of major breakthrough results which almost tightly characterize the gap between the deterministic and randomized complexities for *distributed computation* (in the LOCAL model). Rohzoň and Ghaffari [31] settled a several-decades-old open problems by presenting a deterministic polylogarithmic algorithm for network decomposition. Their result implies that any polylogarithmic-time randomized algorithm for LCL problems [9, 28] can be derandomized to a polylogarithmic-time deterministic algorithm. In other words, *randomization does not help in the polylogarithmic-time regime.* On the other hand, Chang, Kopelowitz and Pettie [9] showed that *in the sub-logarithmic-time regime, randomization might provide an exponential benefit.* For example, for Δ-coloring of trees of maximum degree Δ there are $\Theta(\log_\Delta \log n)$ randomized-round algorithms and a deterministic lower bound of $\Omega(\log_\Delta n)$ rounds. Balliu et al. [4] recently showed that maximal matching and maximal independent sets cannot be found by $o(\Delta + \log\log n/\log\log\log n)$-rounds randomized algorithms, and deterministically in $o(\Delta+\log n)$ rounds, thus provide an exponential gap in the lower bound for bounded-degree graphs. Finally, [8] presented additional separation results for edge-coloring: $(2\Delta-2)$-edge-coloring requires $\Omega(\log_\Delta \log n)$ randomized rounds, and $\Omega(\log_\Delta n)$ deterministic rounds. In view of these separation results, we therefore ask in the context of related MPC computation:

> Question 4. *Is there a gap between component-stable randomized algorithms vs. component-stable deterministic algorithms?*

In this paper, we answer all four questions in the affirmative.

1.2 Our contributions

Our main contribution is in demonstrating the impact of the component-stability property on the complexity of randomized and deterministic local graph problems.

A robust lifting framework. We rectify the framework of low-space component-stable MPC algorithms due to Ghaffari et al. [18]. We study the framework in detail and demonstrate that in order to be broadly applicable, many aspects of the original setting are highly sensitive to the exact definition used, and require amendments to carefully specify what information the outputs of component-stable algorithms may depend on. We present a modified framework, reaching a revised definition of component-stability (see Definition 13) which both encompasses many existing MPC algorithms, and for which robust conditional lower bounds can be shown. This answers Question 1.

Extensions to deterministic and degree-dependent lower bounds. Our revised framework not only recovers all main results from the framework of Ghaffari et al. [18], but also extends the arguments to include conditional lower bounds for *deterministic algorithms* and fine-grained lower bounds that depend on Δ. While our main theorem (Theorem 14) lifting LOCAL lower bounds to component-stable MPC algorithms has several subtle assumptions, the main, informal claim is that for many graph problems $\mathcal{P}$, if $\mathcal{P}$ has a $T(n,\Delta)$-round (randomized or deterministic) lower bound in the LOCAL model, then assuming the connectivity conjecture, any low-space component-stable (respectively, randomized or deterministic) MPC algorithm solving $\mathcal{P}$ requires $\Omega(\log T(n,\Delta))$ rounds.

Instability helps randomized *MPC algorithms.* To address Question 2, we consider the problem of finding a large (specifically, of size $\Omega(n/\Delta)$ independent set. This problem has been recently studied by Kawarabayashi et al. [24], who provided a randomized lower bound of $\Omega(\log^* n)$ rounds (for a specific range of Δ). We show that their lower bound can be adapted to our revised lower-bound lifting framework of Theorem 14, obtaining a conditional lower bound of $\Omega(\log\log^* n)$ rounds for component-stable MPC algorithms.[1] In contrast, we present a very simple $O(1)$-round component-unstable randomized algorithm for the problem. In fact, this algorithm can further be *derandomized* within $O(1)$ rounds (see Theorem 26), demonstrating an instance in which deterministic component-unstable algorithms are more powerful even than randomized component-stable algorithms.

Theorem 5. *Conditioned on the connectivity conjecture, any component-stable low-space MPC algorithm for computing an independent set of size $\Omega(n/\Delta)$ on n-node graphs (for the full range of $\Delta \in [1, n)$) and succeeding with probability at least $1 - \frac{1}{n}$, requires $\Omega(\log\log^* n)$ rounds. This problem admits a simple $O(1)$-round randomized low-space MPC algorithm which is component-unstable; additionally, the algorithm can be derandomized within $O(1)$ rounds.*

The basic observation providing this separation is the fact that one can easily compute in $O(1)$ rounds (even in the LOCAL model) an independent set of $\Omega(n/\Delta)$ nodes *in expectation*. In the LOCAL model, we need provably longer to achieve a high-probability success guarantee of $1 - \frac{1}{n}$. In the low-space MPC model, however, we can perform the process of *success probabiliy amplification*: we run $\Theta(\log n)$ *parallel* repetitions of the basic algorithm, and choose a successful one if such exists, amplifying the success probability to $1 - \frac{1}{n}$ without any slow-down. This powerful process, though, is inherently component-unstable, since it relies on globally agreeing on one of the repetitions to use[2].

Instability helps deterministic *MPC algorithms.* We then turn to consider the effect of component-stability on deterministic MPC algorithms. While the original setup of Ghaffari et al. [18] had been designed only for randomized algorithms, the revised framework developed in our paper in Section 2.4.3 extends naturally to the deterministic setting, providing a robust deterministic lifting analog in Theorem 14. Theorem 14 provides a general framework lifting unconditional deterministic lower bounds for the LOCAL model for many natural graph problems to conditional lower bounds for low-space component-stable MPC algorithms in the same way as the randomized framework in [18].

We then turn to show that with component-instability one can in fact surpass these conditional lower bounds and present several results showing a *separation between deterministic stable and unstable algorithms* (conditioned on the connectivity conjecture) and positively answering Question 3. In Section 3.1, we show that for several problems closely related to LLL, including sinkless orientation and some variants of edge-coloring and vertex-coloring, component-instability helps for deterministic algorithms. Finally, in Section 3.2, we demonstrate a similar result for the class of all LOCAL *extendable* algorithms by combining the lifting of deterministic LOCAL lower bounds in Theorem 14 with a derandomization technique using pseudorandom generators. To demonstrate the applicability of this derandomization recipe, we show how it can be used to *improve the deterministic running times* of two cornerstone problems in low-space MPC: *maximal independent set* and *maximal matching*. And so, on one hand we prove (Theorem 24) that conditioned on the connectivity conjecture, there is no deterministic low-space component-stable MPC algorithm that computes a maximal matching or maximal independent set, even in forests, in $o(\log\Delta + \log\log n)$ rounds, and on the other hand, we give a deterministic low-space component-unstable MPC algorithm for these problem running in $O(\log\log\Delta + \log\log\log n)$ rounds (when $\Delta = 2^{\log^{o(1)} n}$, Corollary 23). (The resulting MPC algorithm must either perform heavy local computations, or alternatively, the underlying PRGs can be hard-coded in the MPC machines for a non-uniform but computationally-efficient algorithm.)

Relations between randomized and deterministic MPC algorithms. Finally, we consider the interesting gap between randomized and deterministic algorithms in the low-space MPC setting. As observed by [9] and [17], randomized algorithms that succeed with probability of $1 - 1/2^{n^2}$ can be turned into non-uniform deterministic algorithms. This result can also be extended to the low-space MPC setting, with some caveat. In contrast to the LOCAL model where the space of the nodes is unlimited, in the low-space MPC setting, the transformation implied by [9] and [17] yields a non-uniform, non-explicit algorithm. By using success probability amplification with $\text{poly}(n)$ machines, one can boost the success guarantee of any randomized MPC algorithm from $1-1/\text{poly}(n)$ to $1-1/2^{n^2}$ without any slowdown in the round complexity. For some specific problems, we can perform more careful derandomization methods that *do not* cause the resulting deterministic algorithms to be non-uniform, non-explicit, or to use excessive global space, demonstrating that component-stability restricts power even without these allowances.

Turning our focus to low-space component-stable MPC algorithms, here we provide a conditional separation between randomized and deterministic algorithms (Theorem 15), positively answering Question 4. This separation follows by combining (i) the conditional lifting for randomized component-stable algorithms and deterministic component-stable algorithm with (ii) problems for which there is provable gap in their randomized and deterministic LOCAL complexity.

Complexity summary: Let us summarize the complexity results, assuming the connectivity conjecture, and allowing non-uniform MPC algorithms. Our study demonstrates that in low-space MPC, component-unstable algorithms are provably stronger than their component-stable counterparts, both for deterministic and randomized algorithms. Further, for component-stable algorithms, randomized algorithms are provably stronger than their deterministic counterparts. However, for arbitrary (possibly component-unstable) algorithms this is not the case: any randomized algorithm can be efficiently simulated by a deterministic one.

[1] The original lifting arguments of [18] only hold for LOCAL lower bounds that hold under exact knowledge of n; the lower bound of [24] does not, but holds under knowing a polynomially-loose estimate of n, which is allowed for in our framework.

[2] Indeed, this causes an issue with the proof of Lemma III.1 of [18], where success probability amplification is used in an algorithm A_{MPC} that is later (in Lemma IV.2 in [18]) assumed to be component-stable.

2 PRELIMINARIES

In this section we suggest an array of changes that may be made to the framework of component-stable algorithms due to Ghaffari et al. [18], in order to reach a revised definition of component-stability (Definition 13) which both encompasses many existing randomized MPC algorithms, and for which robust conditional lower bounds can be shown. These changes also allow us to extend the original setting to both deterministic algorithms and those that have running-time dependency on maximum degree Δ.

2.1 Discussion of component stability

Let us first present the description of component stability from [18, Section II]:

> *Formally, assume that for a graph G, $\mathcal{D}_G$ denotes the initial distribution of the edges of G among the M machines and the assignment of unique IDs to the nodes of G. For a subgraph H of G let $\mathcal{D}_H$ be defined as $\mathcal{D}_G$ restricted to the nodes and edges of H. Let H_v be the connected component of node v. An MPC algorithm $\mathcal{A}$ is called* component-stable *if for each node $v \in V$, the output of v depends (deterministically) on the node v itself, the initial distribution and ID assignment $\mathcal{D}_{H_v}$ of the connected component H_v of v, and on the shared randomness $\mathcal{S}_M$.*

We informally sketch the line of argument of [18] that leverages this definition to lift LOCAL lower bounds to MPC: first, it is shown that if there is a LOCAL lower bound for a problem P, and an MPC algorithm A_{MPC} is able to solve P faster than the log of the LOCAL lower bound, there there must exist two graphs G and G', which are locally indistinguishable but on which A_{MPC}'s output must differ (at least with some sufficiently large probability). In the terminology of [18], A_{MPC} must be 'farsighted': able to somehow make use of information from far-away nodes.

These graphs G and G', and the assumed algorithm A_{MPC}, are then ingeniously used to construct an algorithm $B_{st-conn}$ that solves a connectivity problem conjectured to be hard. Specifically, $B_{st-conn}$ constructs a pair of simulation graphs based on its input to the connectivity problem. These simulation graphs consist of many disjoint copies of induced subgraphs of G and G' respectively. The construction is made in such a way that a *full* copy of G and G' only appears if two particular nodes (designated s and t) are connected in the input graph for the connectivity problem.

$B_{st-conn}$ simulates A_{MPC} on this pair of simulation graphs. If s and t are connected, then full copies of G and G' are present as connected components in the simulation graphs, and A_{MPC} should return *different* outputs on them with sufficiently high probability. Otherwise, there are no full copies of G and G', and A_{MPC} returns the same outputs on both simulation graphs. This difference in behavior is exploited to allow $B_{st-conn}$ to determine whether s and t are connected, and solve the connectivity problem faster than is conjectured to be possible.

The property of component-stability is crucial in this last step: we require that A_{MPC} behaves the same on G and G' when they are connected components of the (much larger) simulation graphs as it does when they are the entire input (as when showing that A_{MPC} was farsighted). Otherwise, we could not say anything about A_{MPC}'s output on the simulation graphs. It transpires that this argument is quite fragile, and highly sensitive to the precise definition of component-stability used. We discuss some of the issues below.

Randomized component-stable algorithms must be allowed dependency on n. The first major point of consideration is that, as defined in [18], the output of a node v under a component-stable algorithm must depend only on shared randomness, the IDs of v and its component, and the input distribution of edges to machines. In particular, no provision is made for dependency on the number of nodes n in the input graph, and indeed, the arguments of [18] seem to forbid it.[3] This is somewhat counter-intuitive for MPC algorithms: while a LOCAL algorithm can never determine n unless it is given as input (and therefore it is commonly assumed that we provide the algorithm with at least a polynomial estimate of n), an MPC algorithm can easily do so in $O(1)$ rounds, by simply summing counts of the number of nodes held on each machine. We can therefore assume any such algorithm *has* knowledge of the exact value of n, and natural algorithmic approaches would generally make use of this knowledge.

Furthermore, the *success* probability of correct randomized algorithms is defined to be at least $1 - \frac{1}{n}$, in accordance with the standard definition of *with high probability correctness*. This causes a contradiction for algorithms with no output dependency on n: consider a correct component-stable MPC algorithm A for a problem in which the validity of a node's output can be verified by seeing its connected component (we will formalize this notion later), running on a n-node graph. This algorithm must produce a valid output for each node in the graph with probability at least $1 - \frac{1}{n}$.

We now add η disconnected nodes to the input graph. If A's output does not have any dependency on n, then it must be unchanged at each of the original nodes, since they are in an unchanged connected component. However, A must succeed on the new graph with probability at least $1 - \frac{1}{n+\eta}$. Since the problem is component-stable, the probability that A succeeds on all of the nodes of the original graph is at least as high as the probability that it succeeds on the whole graph, i.e., $1 - \frac{1}{n+\eta}$. So, A must succeed on the original graph with probability at least $1 - \frac{1}{n+\eta}$, and since we can set η arbitrarily high, with certainty; this requires A to be deterministic!

Another problem with disallowing dependency on n is that *running times* generally depend on n. While, for some (deterministic) algorithms, this dependency could be replaced by other parameters (e.g., the maximum number of nodes in a connected component, or size of the ID space), this is not the case for randomized algorithms, where the running time usually directly affects the success probability, which itself must depend on n.

So, in short, a definition of component-stability which does not allow any dependency on n includes essentially no non-trivial randomized algorithms.

If we allow dependency on n, we must restrict the class of problems in order to obtain MPC lower bounds. We have seen that, to give results which apply to probabilistic algorithms, we must allow dependency on n. However, we cannot then hope to obtain a result akin to Theorem I.4 of [18] for *all* graph problems.

As an example, consider the following problem: each node must output **YES** if the entire graph is a simple path with consecutive

[3] Specifically in proof of [18, Lemma IV.2], where algorithm A_{MPC} is simulated on large simulation graphs containing smaller components G and G', as discussed above, its behavior on G and G' as components is only identical to when run on them as sole input if no dependency on n is permitted.

node IDs, and **NO** otherwise. Note that there is only one possible correct output for each node v, and that this output is a deterministic function of its component and the value of n (since v's output should be **YES** iff its component is an n-node path with consecutive IDs). Furthermore, there is an $O(1)$-round MPC algorithm for the problem: it is straightforward to check whether there are two nodes of degree 1, $n-2$ nodes of degree 2, and that each node's 1-hop neighborhood is consistent with being in a path of consecutive IDs. So, if component-stability allows dependency on n, an $O(1)$-round deterministic component-stable algorithm for the problem exists.

However, the problem has a trivial $n-1$-round (randomized) LOCAL lower bound, since a **YES** instance can be changed to a **NO** instance by only altering the ID of one endpoint of the path, and the other endpoint requires $n-1$ rounds to detect this change. Hence, we cannot hope for a universal method of lifting LOCAL lower bounds to non-trivial component-stable MPC lower bounds if component-stability allows dependency on n.

We will see, though, that such counterexamples are necessarily quite contrived, and that we *can* prove such a result for a class that includes most problems of interest (such as, e.g., *all locally-checkable (LCL)* problems, see Section 2.3).

Uniqueness of identifiers. It is common in both LOCAL and MPC to assume that nodes of the input graph are equipped with identifiers (IDs) that are unique throughout the entire graph. This assumption, however, is somewhat at odds with the concept of component-stability: if, for example, a disconnected node is added to a valid graph, sharing an ID with an existing node, then the input becomes invalid. So, outputs for the original nodes are now allowed to change arbitrarily, even though their components have not altered.

We could instead require that IDs are only component-unique (i.e., they are allowed to be shared by disconnected nodes, but not connected ones). This is a weaker assumption which aligns well with component-stability, and is still sufficient for LOCAL algorithms (where nodes have no interaction with disconnected nodes). This approach, though, presents a problem in MPC. Unlike in LOCAL, where nodes are inherently separate computational entities which only need IDs for symmetry-breaking (particularly for deterministic or shared randomness algorithms), in MPC an input graph node essentially *is* its ID. The input is given only as a binary encoding of the IDs of nodes and edges, and so any two nodes with the same ID will be contracted to a single node when this input is interpreted as a graph. As a consequence, MPC algorithms cannot work with graphs in which IDs are only component-unique.

Our solution to this problem is to separate the two functions of IDs. We assume that IDs are only component-unique, and that component-stable MPC algorithms can depend on these. However, we also provide MPC algorithms with fully-unique *names* for nodes, whose purpose is *only* to allow the algorithm to distinguish the input graph's nodes apart. Accordingly, we do not allow the output of component-stable algorithms to depend on names.[4]

Initial distribution of input. The definition of [18] allows MPC algorithms' outputs to depend on the initial distribution of the input to the machines. While this is natural to do, we observe that under our definition it is not necessary: given a component-stable algorithm A_{MPC} whose output depends on this distribution, we can always create a new component-stable B_{MPC} which does not.

Specifically, since the nodes have unique $poly(n)$ names (and we can also give edges unique $poly(n)$ names based on their endpoints), and we are allowed any $poly(n)$ number of machines, algorithm B_{MPC} can first (in one round) redistribute each node and edge of the input to its own dedicated machine, with the same name as the corresponding node or edge. Then, it simulates A_{MPC}, and reaches a valid output, which is now independent of the initial distribution. Since A_{MPC}'s output is component-stable, B_{MPC}'s is also.[5]

Therefore, a lower bound for component-stable algorithms that depend on input distribution implies a lower bound for those that do not. So, we can disallow this dependency from the definition without weakening our results.

2.2 Graph families

In this section, we make some definitions concerning the input graphs on which MPC algorithms run. Firstly, to address the problem concerning uniqueness of identifiers, we define *legal graphs* to be those with separate unique node names and component-unique node IDs as discussed:

Definition 6. *A graph G is called **legal** if it is equipped with functions ID, name : $V(G) \to [poly(n)]$ providing nodes with IDs and names, such that all names are fully unique and all IDs are unique in every connected component.*

Throughout the paper, we will always assume that input graphs for MPC are legal (and *ensure it when constructing inputs ourselves*). For component-stable algorithms, this is to allow a weaker dependency on the IDs and not the names, as discussed above. For non-component-stable algorithms, it is no different from the standard (fully-unique IDs) requirement, since outputs are allowed to depend on the names, and so we can simply use the names as IDs.

Next, we make a definition which will allow us to show MPC lower bounds on specific families of graphs. LOCAL lower bounds are often proven using graphs from some specific family $\mathcal{H}$ as the *"hard instances"*: in particular, many such bounds are proven on trees. Lower bounds on restricted families of graphs are stronger than those on the class of all graphs, and can also provide meaningful hardness results for problems which are only *possible* on restricted families (such as Δ-vertex coloring, see Theorem 20). When lifting LOCAL lower bounds on restricted graph families, we therefore wish to preserve the family on which the lower bound holds. We find that families satisfying the following property can be preserved (see the full version for discussion):

Definition 7 (Normal families of graphs). *A class of graphs $\mathcal{H}$ is called **normal** if it is hereditary (i.e., closed under removal of nodes) and closed under disjoint union.*

[4]Note that, unlike the changes regarding dependency on n and problem class, this change is not necessary to show a general framework for conditional MPC lower bounds — the same results could be proven assuming fully-unique IDs (at least for randomized algorithms) using techniques from [18]. However, we feel that this definition better captures the 'spirit' of component-stability.

[5]We will define component-stable outputs to not depend on the names of machines — this is not a major restriction, since we are not aware of any MPC algorithms which are not independent of renaming machines. However, it is an important point here, since B_{MPC}'s machine names now depend on node names, upon which B_{MPC}'s output must not depend.

The set of *all graphs* is a normal family, and can always be used in the worst case. Further, observe that the family of *all trees* is *not* normal; however, the family of *all forests* is normal. So, for example, Theorem 14 implies that LOCAL lower bounds on *trees* can be lifted to conditional MPC lower bounds on *forests* (but not trees).

2.3 Types of graph problems and replicability

We next define the types of *problem* we will encompass with this work. We will focus on graph problems, and let $\mathbb{G}$ be the collection of all legal input graph instances.

We consider only graph problems where *each node of the input graph must output some label* from a finite set Σ. For example, for the vertex coloring problem the label of a node corresponds to its color, and for the independent set problem, the label corresponds to the indicator variable whether the node is in the independent set returned. A straightforward reduction using line graphs allows us to apply the framework also to *edge-labelling* problems.

A graph problem is then defined by a collection of *valid* outputs for each possible pair (topology, IDs) of a legal input graph. Importantly, we do *not* allow validity to be dependent on the *names* of graph nodes (though these names are part of any legal input). That is, given a particular input graph topology and set of IDs, the collection of valid outputs must be consistent regardless of node names. This is because component-stable outputs are not allowed to depend on names, so most problems which allowed solution-dependency on names would be trivially unsolvable by component-stable algorithms. In any case, names were introduced solely to allow MPC algorithms to distinguish nodes as objects, and should not be considered part of problems.

The goal of an algorithm is then to provide a valid output for the specific legal input it was given. For many problems it is useful to have a concept of the output of a *particular node* being valid. The overall output is then valid if all nodes' outputs are valid. To capture this concept, we define the following sub-class of problems:

Definition 8. *For $r \in \mathbb{N}$, an r-**radius checkable problem** is defined by a collection of valid outputs for each r-radius centered graph equipped with unique IDs.*[6] *The output of a node v in input graph G is deemed valid if the centered graph given by its r-radius ball, and the outputs thereof, is a member of this valid collection. An overall output on G is valid if all nodes' outputs are valid.*

An *r-radius centered graph* here is simply a connected graph with a designated center node, from which all other nodes are of distance at most r.

One can see that r-radius checkable problems are precisely those whose solutions can be verified in r rounds of LOCAL. Note that the majority of graph problems of interest are r-radius-checkable for some $r \leq n$: for example, the vertex coloring problem requires that each node outputs a color distinct from the colors output by its neighboring nodes, and thus is easily 1-radius-checkable. Similarly, all LCL (*locally-checkable labeling*, see, e.g., [9, 28]) problems, a commonly studied class particularly from a lower-bounds perspective, are $O(1)$-radius checkable. Still, some natural problems are not n-radius checkable problems: most notably, approximation problems are not, since there is no notion of a node's validity, and nor can nodes determine overall validity by seeing their n-radius ball (i.e., their entire connected component). So, while some of our results concern r-radius checkable problems (such as those in Section 3.2), our main lower bounds results will use a more general class of problems, see below, in order to incorporate approximation problems.

2.3.1 Replicable graph problems. We have seen, from Section 2.1, that to transfer LOCAL lower bounds to MPC, under a definition of component-stability that includes randomized algorithms (and so allows dependency on n), one must restrict the class of problems, since some (contrived) problems have $\Omega(n)$-round LOCAL lower bounds and $O(1)$-round MPC algorithms. Our goal in this section is to make the minimal restriction needed to facilitate such lower-bound lifting arguments.

During proof of Theorem 14 (our main lower-bound lifting theorem), we will consider multiple disjoint copies of the input graph enhanced by isolated nodes. To facilitate this concept in our analysis, we introduce the notion of *replicable graph problems*.

Definition 9. *A graph problem is R-**replicable** if it satisfies the following property. For any*

- *graph $G \in \mathbb{G}$ with $|V(G)| \geq 2$,*
- *output labeling $L : V(G) \to \Sigma$,*
- *individual output label $\ell \in \Sigma$, and*
- *graph Γ_G which is a disjoint union of at least $|V(G)|^R$ disjoint copies of G (with the same IDs as G) and fewer than $|V(G)|$ isolated nodes (with the same ID as each other),*

let output labeling L' on Γ_G be given by L on each copy of G, and ℓ on each isolated node. Then, if L' is valid on Γ_G, L must be valid on G.

The definition of replicability may seem unnatural: it is designed to align with a specific construction during proof of Theorem 14. However, we argue that the vast majority of natural graph problems are replicable. We first show all that r-radius-checkable problems (and hence all LCL problems [9, 28]) are replicable:

Lemma 10. *Any r-radius-checkable problem is 0-replicable.*

Further, a major strength of our framework is that *most approximation problems are also replicable*. As an example, we show replicability for the problem of finding an independent set of size $\Omega(n/\Delta)$, a problem for which we later (in Section 4) show a separation between component-stable and non-component-stable algorithms.

Lemma 11. *The problem of finding an independent set of size $\Omega(n/\Delta)$ (on graphs with $\Delta \geq 1$) is 2-replicable.*

Similarly, we have a related lemma for approximate matching, one of the central problems demonstrating the power of our framework summarized in Theorem 14 (which will yield also the conditional hardness of the approximate maximum matching problem on MPC). The same arguments can also straightforwardly show that $\Omega(1)$-approximation of maximum matching and minimum vertex cover are $O(1)$-replicable.

Lemma 12. *The problem of finding an $\Omega(1)$-approximation of maximum matching is 2-replicable.*

[6] r-radius graphs are, by definition, connected, so component-unique IDs are unique IDs.

2.4 Algorithm definitions, and revised definition of component-stability

Once we have defined the type of *problems* we consider, we may define LOCAL and MPC algorithms that solve them, and in particular, give a formal, amended definition of component-stable algorithms used in this paper, taking into account the discussion above.

2.4.1 LOCAL algorithms. Our formal definition of algorithms in the LOCAL model used in this paper is as follows:

Input. LOCAL algorithms receive as input an n-node graph G, with component-unique IDs for each node. Randomized algorithms also provide each node with access to a *shared, unbounded, random seed* $\mathcal{S}$. Algorithms are provided with the exact value of the maximum degree Δ, and an *input size estimate* N of n such that $n \leq N \leq poly(n)$.[7]

The nodes of the input graph are the computational entities, and each initially has knowledge of its adjacent edges in G (i.e., the IDs of their other endpoints). The computation proceeds in synchronous rounds, and in each round, a node may send an arbitrary message along each of its adjacent edges. At the termination of the algorithm, each node must give an output label from Σ.

Output. Correct deterministic algorithms must provide a valid overall output labeling for the problem, on every output; randomized algorithms must give a valid labeling with probability at least $1 - \frac{1}{N}$, over the distribution of random seed $\mathcal{S}$, for any input.

Shared randomness. Given that MPC algorithms naturally allow shared randomness, it is important for our study of randomized LOCAL algorithms to allow the nodes to have access to shared randomness too. The use of *shared randomness* is non-standard in the LOCAL model, where one typically assumes only private randomness. However, as shown by Ghaffari et al. [18, Section V], many of the existing LOCAL lower bounds can be extended (and without any asymptotic loss in their LOCAL round complexity) also if the nodes have access to shared randomness. (Notice that the notion of shared randomness is only relevant to randomized algorithms; for deterministic complexity one can use the existing deterministic LOCAL lower bounds as black box results.)

2.4.2 MPC algorithms. We use the standard definition of MPC algorithms (see, e.g., [2, 6, 13, 16, 20, 23]) amended to fit the framework of low-space MPCs used in the paper.

Input. MPC algorithms receive as input a legal n-node graph G, distributed arbitrarily over $poly(n)$ machines, each with local space $O(n^\phi)$ for some $\phi < 1$. Randomized algorithms also provide each node with access to a shared, random seed $\mathcal{S}$ of $poly(n)$ bits (again distributed arbitrarily among machines). We do not assume that Δ or n are given explicitly as input, but MPC algorithms can determine them easily in $O(1)$ rounds, so we may assume knowledge thereof.

Computation proceeds in synchronous rounds, and in each round, a machine first perform an arbitrary local computations on its local data and then may send and receive a total of $O(n^\phi)$ information, divided between any other machines as desired. At the termination of the algorithm, each machine must give an output label from Σ for each node it received in the initial distribution.

Output. Correct deterministic algorithms must always provide a valid overall output labeling for the problem, on every output; randomized algorithms must give a valid labeling with probability at least $1 - \frac{1}{n}$, over the distribution of random seed $\mathcal{S}$, for any input.

Computation in MPC algorithms. While we are mainly using the most standard setup of MPC algorithms, closely following, e.g., [2, 6, 13, 16, 20, 23], occasionally we will use some features which (while often standard) are less commonly used.

The standard MPC model assumes that in each synchronous rounds, each machine performs arbitrary local computations on its data (fitting its local memory of size $S = O(n^\phi)$) and then the machines simultaneously exchange messages, in a way that each machine is sender and receiver of up to $O(S)$ messages. While some papers also consider the sequential running time of any single MPC machine in every round, the main focus of our study is primarily on the information theoretic aspects of understanding the round complexity in MPC algorithms. (Notice that unbounded local computation assumption is standard in the classical distributed models as LOCAL, CONGEST, and CONGESTED CLIQUE.) As a result, while many of our algorithms perform only poly(n)-time computations, occasionally we will allow MPC machines to perform *heavy local computations*, up to $2^{O(S)}$ local computations in a round; still, the space used on a single machine remains $S = O(n^\phi)$. Our results show that allowing such heavy computations might provide advantageous in the context of deterministic algorithms and derandomization, however they are not necessary to find examples of component-unstable deterministic algorithms which surpass component-stable conditional lower bounds.

Furthermore, while typically one is concerned with the design of uniform MPC algorithms, as it has been observed by Fish et al. [15], the original setup of MPC (e.g., [23]) leads naturally to the non-uniform model of computation. Most of the MPC algorithms presented in our paper are uniform, but occasionally we use *non-uniform algorithms*. In our setting, this means that the MPC algorithm, on each single machine initially knows the number of nodes n (or its estimation), and possibly different algorithms are used for different values of n. This can be also seen as having some non-uniform advice hardwired in the algorithms on individual MPC machines (or as Boolean circuits; for more details, see, e.g., Section 7.1.1 in [32]). Some of these non-uniform algorithms we use are also *non-explicit*. That is, we will be showing that there is a low-space MPC algorithm for a specific task, but we will not provide a procedure to explicitly design it (a brute-force procedure is generally obvious from the proof, but requires exponential computation, and possibly also too much space to perform in low-space MPC). In this paper, non-uniform, non-explicit MPC algorithms will be occasionally used in the context of derandomization.

2.4.3 Component-stable MPC algorithms. Now, after our discussion in Sections 2.1 to 2.3, we are ready to provide a new definition of component-stable MPC algorithms used in this paper.

Definition 13 (**Component-stable MPC algorithms**). *A randomized MPC algorithm A_{MPC} is **component-stable** if its output*

[7] The reason we assume that only a polynomial estimate N of n is known here is that some LOCAL lower bounds to which we wish to apply our lifting result only hold without exact knowledge of n (e.g., that of [24]). Most LOCAL lower bounds, however, do hold under exact knowledge, and in these cases we can simply set $N = n$.

at any node v is entirely, deterministically, dependent on the topology and IDs (but independent of names) *of v's connected component (which we will denote $CC(v)$), v itself, the exact number of nodes n and maximum degree Δ in the* entire *input graph, and the input random seed $\mathcal{S}$. That is, the output of A_{MPC} at v can be expressed as a deterministic function $A_{MPC}(CC(v), v, n, \Delta, \mathcal{S})$.*

A deterministic MPC algorithm A_{MPC} is component-stable under the same definition, but omitting dependency on the random seed $\mathcal{S}$.

Finally, let us state the main technical result demonstrating the power of our revised framework of component-stable MPC algorithms, lifting unconditional lower bounds from the LOCAL model to conditional lower bounds for low-space component-stable MPC algorithms. The following theorem extends the main result in the component-stable algorithms framework due to Ghaffari et al. [18, Theorem I.4] to our framework and enhances it to include lower bounds against *deterministic algorithms*, and with *dependency on maximum input degree* Δ. Informally, similarly to [18, Theorem I.4], Theorem 14 below states that, conditioned on the connectivity conjecture, for $O(1)$-replicable graph problems, any $T(N, \Delta)$-round lower bound in the LOCAL model yields a $\Omega(\log T(N, \Delta))$-round lower bound for any component-stable low-space MPC algorithm. Furthermore, the claim holds for both randomized and deterministic algorithms (deterministic algorithms were not studied in [18]).

Theorem 14 (**Lifting LOCAL lower bounds to component-stable MPC algorithms**). *Let $\mathcal{P}$ be a $O(1)$-replicable graph problem that has a $T(N, \Delta)$-round lower bound in the randomized LOCAL model with shared randomness, for constrained function T, on graphs with input estimate N and maximum degree Δ, from some normal family $\mathcal{G}$. Suppose that there is a randomized $o(\log T(n, \Delta))$-round low-space component-stable MPC algorithm $\mathcal{A}_{MPC}$ for solving $\mathcal{P}$ on legal n-node graphs with maximum degree Δ from $\mathcal{G}$, succeeding with probability at least $1 - \frac{1}{n}$. Then, there exists a low-space randomized MPC algorithm $\mathcal{A}^*$ that can distinguish one n-node cycle from two $\frac{n}{2}$-node cycles in $o(\log n)$ rounds, succeeding with probability at least $1 - \frac{1}{n}$.*

The same holds if the LOCAL lower bound and algorithm $\mathcal{A}_{MPC}$ are both deterministic (but the resulting $\mathcal{A}^*$ remains randomized).

(The notion of a *constrained function* is defined precisely in the full version, but can be thought of here as any "reasonable" function that grows slower than logarithmically in N).

3 SEPARATION BETWEEN STABLE AND UNSTABLE DETERMINISTIC MPC

We start by presenting a general statement for characterization of the graph family of local problems for which component-unstable MPC algorithms provably help. Specifically, this includes problems with a provable exponential gap between their LOCAL deterministic and randomized complexities (e.g., as shown in [4, 8, 9]).

Theorem 15. *Let $\mathcal{P}$ be a $O(1)$-replicable graph problem. Let $T_r(N)$ and $T_d(N)$ be the randomized and deterministic, respectively, LOCAL round complexity of $\mathcal{P}$ on bounded-degree graphs with n nodes, with an input size estimate N and exact knowledge of Δ. If $T_r(N) < \log^\gamma N$ for some constant $\gamma \in (0, 1)$, and if there is a provable exponential gap between $T_r(N)$ and $T_d(N)$, then, assuming the connectivity conjecture, there is an exponential gap between component-stable and component-unstable deterministic low-space MPC complexities for solving $\mathcal{P}$.*

Notice that Theorem 15 demonstrates that, assuming the connectivity conjecture, there are some graph problems for which there are deterministic low-space component-unstable MPC algorithms that are provably faster than their component-stable counterparts. However, the weakness of Theorem 15 is that the obtained deterministic low-space component-unstable MPC algorithms are *non-uniform*. In the rest of this section we will address this issue and show that a similar claim holds also for uniform deterministic MPC algorithms, and those which use almost-optimal global space.

3.1 Lovász Local Lemma-related problems

We first demonstrate a deterministic complexity separation between component-stable and component-unstable algorithms for a group of problems related to the distributed Lovász Local Lemma: sinkless orientation, $(\Delta + o(\Delta))$-edge coloring, and $o(\Delta)$-coloring triangle-free graphs. These problems are known to be hard in the LOCAL model via proofs based on the *round elimination* technique [8]. We show that by derandomizing an algorithm for the constructive Lovász Local Lemma and plugging this into known algorithms for the problems, we can surpass the component-stable lower bounds we obtain when lifting the LOCAL lower bounds to low-space MPC.

3.1.1 Sinkless orientation. We define sinkless orientation to be the problem of orienting the edges of a graph, such that each node of minimum degree at least 3 has at least one outgoing edge (this minimum degree criterion is necessary, since otherwise the problem is not possible, e.g., on a path). A lower bound for sinkless orientation in the LOCAL model was first proven by [7], and extended to a stronger bound for deterministic algorithms by [9]. When combined with Theorem 14, we obtain the following theorem.

Theorem 16. *Assuming the connectivity conjecture, there is no deterministic component-stable low-space MPC algorithm that computes a sinkless orientation in $o(\log \log_\Delta n)$ rounds, even in forests.*

Theorem 16 is complemented by the following result providing a deterministic low-space component-unstable MPC algorithm for sinkless orientation that surpasses the component-stable lower bound for low-space MPC.

Theorem 17. *There is a deterministic low-space MPC algorithm that computes a sinkless orientation, in any graph of $\Delta = \log^{o(1)} \log n$ maximum degree, in $poly(\Delta) + O(\log \log \log n) = o(\log \log_\Delta n)$ rounds, using $n^{1+o(1)}$ global space. The algorithm is component-unstable, and uses $n^{poly(\Delta)}$ local computations.*

We also remark that, for $\Delta \leq n^{o(1/\log \log n)}$, an $O(\log \log \log n)$-round component-stable *randomized* algorithm exists for the problem, by simply collecting $\Theta(\log \log n)$-radius balls onto machines via graph exponentiation, and then simulating the randomized LOCAL algorithm of [19]. Sinkless orientiation is therefore an example of a problem with a (conditional) separation between randomized and deterministic algorithms.

3.1.2 Edge-coloring. Similarly to the sinkless orientation problem, the framework combining the lifting of the LOCAL lower bounds to

deterministic low-space component-stable MPC algorithms (Theorem 14) with derandomization of the constructive Lovász Local Lemma can be used to show that assuming the connectivity conjecture, for the classical edge-coloring there are deterministic low-space component-unstable MPC algorithms that are provably faster than their component-stable counterparts.

Theorem 18. *Assuming the connectivity conjecture, there is no deterministic component-stable low-space MPC algorithm that computes a $(2\Delta-2)$-edge coloring, even in forests, in $o(\log \log_\Delta n)$ rounds.*

Theorem 19. *There is a deterministic low-space MPC algorithm that computes a $(\Delta+\sqrt{\Delta}\log^3\Delta)$-edge coloring, in any graph of maximum degree $\Delta = \log^{o(1)} \log n$, in $O(poly(\Delta) + \log\Delta \log\log\log n) = o(\log\log_\Delta n)$ rounds, using $n^{1+o(1)}$ global space. The algorithm is component-unstable and uses $n^{poly(\Delta)}$ local computation.*

3.1.3 Vertex-coloring triangle-free graphs. We can also show a similar complexity gap between deterministic component-stable and component-unstable MPC algorithms for vertex coloring in triangle-free graphs (notice that all forests are triangle-free).

Theorem 20. *Assuming the connectivity conjecture, there is no deterministic component-stable low-space MPC algorithm that computes a Δ-vertex coloring, even in forests, in $o(\log\log_\Delta n)$ rounds.*

Theorem 21. *There is a deterministic low-space MPC algorithm that computes a $O(\frac{\Delta}{\log\Delta})$-vertex coloring, in any triangle-free graph of $\Delta = \log^{o(1)} \log n$ maximum degree, in $poly(\Delta)+O(\log\Delta\log\log\log n) = o(\log\log_\Delta n)$ rounds, using $n^{1+o(1)}$ global space. The algorithm is component-unstable and uses $n^{poly(\Delta)}$ local computations.*

For all three problems above, we have obtained component-unstable algorithms which surpass the conditional lower bounds for component-stable algorithms when $\Delta = \log^{o(1)} \log n$. Furthermore, though in general these algorithms use heavy local computation, for bounded degree ($\Delta = O(1)$) graphs their local computation is $\mathrm{poly}(n)$. Since we still surpass the lower bounds for sufficiently large constant Δ, this demonstrates that *component-instability helps for deterministic algorithms even using polynomial computation.*

3.2 Extendable algorithms

We next give an explicit uniform derandomization recipe for a particular class of LOCAL algorithms, that we call *extendable* algorithms. We defer the formal definition to the full version, but roughly speaking, these algorithms can extend any partial legal solution into a complete solution (similar to the notion of *greedy* algorithms). The derandomization recipe allows one to derandomize r-round extendable LOCAL algorithms within $O(\log r)$ low-space MPC rounds, provided that the r-radius ball of each node in the graph G fits the machines' local space. Consequently, we show component-unstable algorithms for maximal independent set and maximal matching that both improve over the previous best deterministic algorithms of [11], and surpass the component-stable lower bounds when $\Delta = 2^{\log^{o(1)} n}$.

Theorem 22. *For any constant $\phi > 0$, a maximal independent set and maximal matching can be found deterministically in low-space MPC in $O(\log\log\Delta + \log\log\log n)$ rounds when $\Delta = 2^{\log^{o(1)} n}$, with local space $O(n^\phi)$ and global space $O(n^{1+\phi})$.*

Corollary 23. *For any constant $\phi > 0$, a maximal independent set and maximal matching can be found deterministically in low-space MPC in $O(\log\Delta + \log\log\log n)$ rounds, with local space n^ϕ and global space $O(n^{1+\phi})$.*

Theorem 24. *Assuming the connectivity conjecture, there is no deterministic low-space component-stable MPC algorithm that computes a maximal matching or maximal independent set in $o(\log\Delta + \log\log n)$ rounds.*

4 SEPARATION BETWEEN STABLE AND UNSTABLE RANDOMIZED MPC

In this section, we demonstrate the existence of a natural problem with a (conditional) gap between *randomized* component-stable and component-unstable algorithms, yielding Theorem 5.

We consider the task of computing large independent sets in n-node graphs with maximum degree Δ. Recently, [24] has shown that for any n, there exists a collection of n-node graphs with maximum degree $\Delta = \Omega(n/\log n)$, for which any randomized LOCAL algorithm for computing an independent set of size $\Omega(n/\Delta)$ with success probability $1 - \frac{1}{n}$ (in fact, even reaching a weaker success guarantee of $1 - \frac{1}{\log n}$), requires $\Omega(\log^* n)$ rounds. (Here, and throughout this section, we assume that $\Delta \geq 1$ so that the problem is well-defined.) We provide a mild adaptation to the lower bound proof of [24] so that it would fit the framework from Section 2 and from [18], yielding the following:

Lemma 25 (Super-constant lower bound for component-stable IS). *Assuming that the connectivity conjecture holds, there is no $o(\log\log^* n)$-round low-space component-stable MPC algorithm that computes an independent set of size $\Omega(n/\Delta)$, in any graph with n nodes and $\Delta \in \Delta \in \{\Theta(n/\log n), \Theta(n/\log^* n)\}$, with success probability at least $1 - \frac{1}{n}$.*

However, we can also show that there is a simple (component-unstable) MPC algorithm for computing large independent sets in a constant number of rounds, which can furthermore be derandomized, still in $O(1)$ rounds.

Theorem 26. *There is a deterministic low-space MPC algorithm that in $O(1)$ rounds computes an independent set of size $\Omega(n/\Delta)$, in any graph on n nodes with $\Delta = [1, n)$.*

5 CONCLUSIONS

In this paper, we investigate the power of component-instability for solving local graph problems in the low-space MPC model. Our main conclusion is that component instability is useful mainly in two (possibly related) aspects: amplification of the success guarantee and derandomization. In the context of randomized algorithms, it allows one to boost the success guarantee of the algorithm. This appears to be useful especially for approximation problems . In the context of derandomization, it allows one to efficiently simulate the randomized local algorithm while globally searching for a short seed. Amplification and derandomization are both obtained by a global computation regardless of the connected components of the graph. A major open problem left by this work is to provide conditional hardness results for graph problems that hold also for component-unstable algorithms.

REFERENCES

[1] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel Algorithms for Geometric Graph Problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 574–583.

[2] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. 2018. Parallel Graph Connectivity in Log Diameter Rounds. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS)*. 674–685.

[3] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. 2019. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1616–1635.

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2019. Lower Bounds for Maximal Matchings and Maximal Independent Sets. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 481–497.

[5] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication Steps for Parallel Query Processing. In *Proceedings of the 32nd ACM SIGMOD Symposium on Principles of Database Systems (PODS)*. 273–284.

[6] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. 2019. Exponentially Faster Massively Parallel Maximal Matching. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 1637–1649.

[7] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. 2016. A Lower Bound for the Distributed Lovász Local Lemma. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 479–488.

[8] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. 2020. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Transactions on Algorithms* 16, 1 (Jan. 2020), 8:1–8:51.

[9] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2019. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. *SIAM Journal on Computing* 48, 1 (2019), 122–143.

[10] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. 2019. The Complexity of $(\Delta+1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*. 471–480.

[11] Artur Czumaj, Peter Davies, and Merav Parter. 2020. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *Proceedings of the 32nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 175–185.

[12] Artur Czumaj, Peter Davies, and Merav Parter. 2020. Simple, Deterministic, Constant-Round Coloring in the Congested Clique. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*. 309–318.

[13] Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round Compression for Parallel Matching Algorithms. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*. 471–484.

[14] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation (OSDI)*. 137–149.

[15] Benjamin Fish, Jeremy Kun, Ádám Dániel Lelkes, Lev Reyzin, and György Turán. 2015. On the Computational Complexity of MapReduce. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*. 1–15.

[16] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*. 129–138.

[17] Mohsen Ghaffari and Fabian Kuhn. 2019. On the use of Randomness in Local Distributed Graph Algorithms. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*. 290–299.

[18] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. 2019. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 1650–1663.

[19] Mohsen Ghaffari and Hsin-Hao Su. 2017. Distributed Degree Splitting, Edge Coloring, and Orientations. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2505–2523.

[20] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1636–1653.

[21] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, Searching, and Simulation in the MapReduce Framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*. 374–383.

[22] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review* 41, 3 (March 2007), 59–72.

[23] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 938–948.

[24] Ken-ichi Kawarabayashi, Seri Khoury, Aaron Schild, and Gregory Schwartzman. 2020. Improved Distributed Approximation to Maximum Independent Set. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC)*. 35:1–35:16.

[25] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2006. The Price of Being Near-sighted. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 980–989.

[26] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: A Method for Solving Graph Problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 85–94.

[27] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing* 21, 1 (Feb. 1992), 193–201.

[28] Moni Naor and Larry J. Stockmeyer. 1995. What Can be Computed Locally? *SIAM Journal on Computing* 24, 6 (1995), 1259–1277.

[29] Krzysztof Onak. 2018. Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space. *CoRR abs/1807.08745* (2018).

[30] Tim Roughgarden, Sergei Vassilvitski, and Joshua R. Wang. 2018. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *Journal of the ACM* 65, 6 (Nov. 2018), 41:1–41:24.

[31] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-time Deterministic Network Decomposition and Distributed Derandomization. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*. 350–363.

[32] Salil P. Vadhan. 2012. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science* 7, 1-3 (2012), 1–336.

[33] Tom White. 2012. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

[34] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.